UTAH GOVPAY 3.0:

THE OFFICIAL PAYMENT SOLUTION FOR UTAH GOVERNMENT

Technical Manual

UtahInteractive

"Extend the power of your website…by taking payments online"

This Utah GovPay Technical Manual contains the following information:

GOVPAY METHOD OPTIONS

Utah GovPay provides a secure transaction processing solution to and is able to process transactions in two different methods. Each of these methods includes different feature sets. The following descriptions are brief overviews for these methods. Detailed feature sets of each method are located later in this document.

1. Post method – This method provides a simplistic approach to payment processing by transmitting payment data via secure URL. It also provides a post back function that will relay transaction status back to the agency's application. This method does not allow for multiple line item payments or customizable data fields.
2. WPS Web service method –This provides a more complex and customizable payment processing method via secure web service. It can be customized by adding additional data fields and multiple line items. It does not initiate the return of transaction status data. Transaction status is obtained only when the agency's application initiates a query for the transaction status data.

POST METHOD OVERVIEW:

The Utah GovPay POST implementation allows for a simple method of passing customer transaction information between the agency's web application and Utah GovPay. The process is initiated by POSTing information using a URL and Utah GovPay will complete the transaction. Once the transaction is complete, Utah GovPay will use a post-back URL to return the transaction status to your agency's web application.

POST TERMINOLOGY

* Post - The process of transmitting data via URL. The calling application will pass this data string to GovPay.
* Calling Application – This is the outside application built by the agency.
* Web Service – A piece of software that can be accessed over the Internet by another application using XML to send or retrieve information.
* Web Application – A web application uses a web site as a front end to interact with users across the Internet.

POST FEATURES

Current features of the Post method are:

* Simple yet effective payment method.
* Register transaction from calling application to GovPay.
* Post transaction back to calling application.

Please use the test URL until you are ready to accept live transactions. If you have any questions please contact your product manager at Utah Interactive for assistance.

POST URL for test:
https://test.secure.utah.gov/govpay/checkout

POST URL for production (to be used only to accept live transactions):
https://secure.utah.gov/govpay/checkout

PostParameters		
Property	**Required**	**Description**
account_name	YES	the name of the account we are going to use to perform this transaction against (this will make sure it ends up in the correct GovPay account).
post_back_url	YES	the URL we will use to post transaction data back to the calling application.
payment_types	YES	CREDITCARD for credit card, ECHECK for e-check. If you utilize both separate them with the | symbol. Example: CREDITCARD|ECHECK.
shared_secret_name	ECHECK	the Shared Secret name for a Electronic check transaction (required only when ECHECK is used)
shared_secret_value	ECHECK	the Shared Secret value for a Electronic check transaction. (required only when ECHECK is used)
amount	YES	the amount for this transaction.
transaction_id	YES	the transaction id you want to give to this transaction.
item	YES	the description of the item you are selling.
name	NO	the name of the person that is going to complete the transaction (if already known beforehand).
address_line_1	NO	the address line 1 of the person that is going to complete the transaction (if already known

		beforehand).
address_line_2	NO	the address line 2 of the person that is going to complete the transaction (if already known beforehand).
city	NO	the city of the person that is going to complete the transaction (if already known beforehand).
postal_code	NO	the postal code of the person that is going to complete the transaction (if already known beforehand).
state	NO	the state/province of the person that is going to complete the transaction (if already known beforehand).
email_address	NO	the email address of the person that is going to complete the transaction (if already known beforehand).
success_url	NO	the URL we will direct to after the POST completes.

Post Example:

```
<html>
   <head><title>GovPay Test Page</title></head>
   <body>
      <form method="POST" action="https://test.secure.utah.gov/checkout">
         <table>
            <tr>
               <td>Post Back URL:</td>
               <td><input type="text" name="post_back_url" value=""></td>
            </tr>
            <tr>
               <td>Amount:</td>
               <td><input type="text" name="amount" value=""></td>
            </tr>
            <tr>
               <td>Transaction ID:</td>
               <td><input type="text" name="transaction_id" value=""></td>
            </tr>
            <tr>
               <td>Description:</td>
```

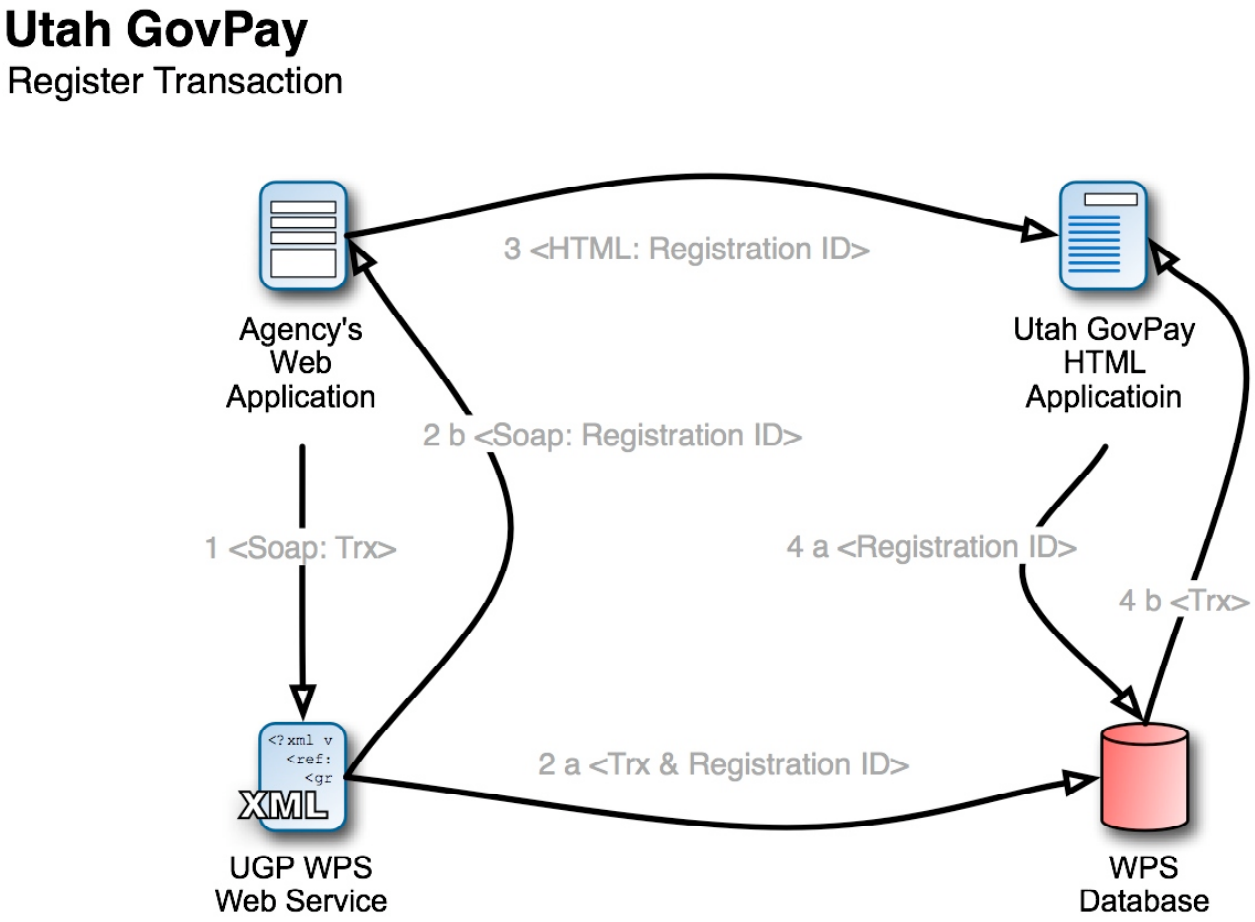
```
            <td><input type="text" name="item" value=""></td>
          </tr>
          <tr>
            <td>Name:</td>
            <td><input type="text" name="name" value=""></td>
          </tr>
          <tr>
            <td>Address line 1:</td>
            <td><input type="text" name="address_line_1" value=""></td>
          </tr>
          <tr>
            <td>Address line 2:</td>
            <td><input type="text" name="address_line_2" value=""></td>
          </tr>
          <tr>
            <td>City: </td>
            <td><input type="text" name="city" value=""></td>
          </tr>
          <tr>
            <td>Postal Code: </td>
            <td><input type="text" name="postal_code" value=""></td>
          </tr>
          <tr>
            <td>State/Province: </td>
            <td><input type="text" name="state" value=""></td>
          </tr>
          <tr>
            <td>Email Address:</td>
            <td><input type="text" name="email_address" value=""></td>
          </tr>
          <tr>
            <td colspan="2" align="right">
              <input type="hidden" name="account_name" value="myaccount">
           <input type="hidden" name="payment_types" value="CREDITCARD|ECHECK">
              <input type="submit" value="Submit">
            </td>
          </tr>
        </table>
      </form>
    </body>
</html>
```

WPS WEB SERVICE METHOD OVERVIEW:

The Utah GovPay WPS web service provides a secure method to pass customer transaction information between the Agency's web application and Utah GovPay via web service.

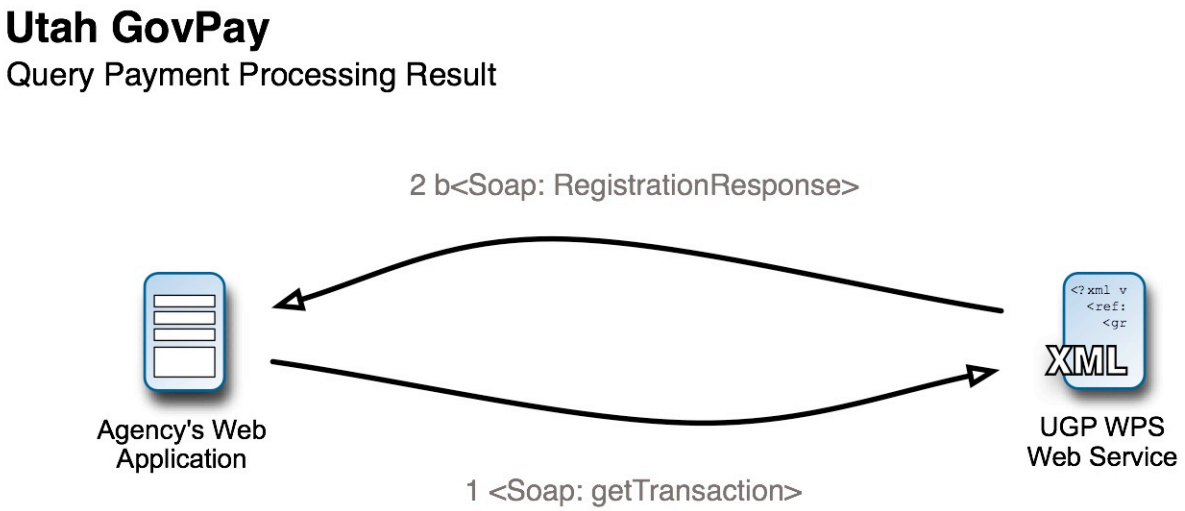
The WPS is the backend link into the Utah GovPay system and was designed to prevent web users from fraudulently altering their own transaction data. WPS has two main functions, registering transactions and querying transactions. In the registration process, the agency's web application sends the transaction data to WPS and WPS returns a registration ID. The Agency's web application then forwards the user to Utah GovPay with the registration ID.

After a completed payment transaction, the Agency's web application can use the registration ID to query WPS to find out if the transaction was approved.



The Register Transaction process follows the following steps:

1. When the user is ready to make a payment, the agency's web application sends details of the payment in XML using the Soap format to the Utah GovPay Web Service or WPS.
2. The Utah GovPay WPS creates a Registration ID, stores the transaction information and registration ID in a database and returns a registration ID back to the agency's web application.
3. The agency's web application redirects the user to the Utah GovPay URL and includes the registration ID in the query string. This Utah GovPay URL will be created during the Utah GovPay setup.
4. The Utah GovPay website uses the registration ID to retrieve the transaction data and then takes the user through the payment process.



The Query Transaction process:

1. The agency's web application sends a soap message with the registration ID
2. The Utah GovPay Web Service returns the results of the transaction. The details of the soap message are listed below in the Complex XML types under **TransactionDetailResponse.**

WPS TERMINOLOGY

* Registration Id - The unique identifier used generated by WPS. The calling application should pass this to WPS when the user is handed off.
* Calling Application – This is the outside application built by the agency.
* Web Service – A piece of software that can be accessed over the Internet by another application using XML to send or retrieve information.
* Web Application – A web application uses a web site as a front end to interact with users across the Internet.

WPS FEATURES

Current features of the WPS web service are:

* Register Transaction Details

* Retrieve TransactionDetailResponse in XML for a single transaction
* Retrieve StatusResponse in XML for a single transaction

OPERATIONS

The following operations are available in the WPS web service.

Register a Transaction – Information about the transaction is sent from the Calling Application to WPS and a registration ID is sent back the Calling Application. The name of this operation is "register".

Query a Transaction – The registration ID or a group of registration ID's are sent from the Calling Application to WPS and the results of the transaction are returned to the Calling Application. There are two operations that can be used to retrieve information about the transaction after it has been processed.
* getTransaction returns all the transaction information in an XML format.
* getStatus returns transaction status information in an XML format.

Operations:

1. **register**

   Register a transaction with WPS using the information in the RegistrationRequest object that was passed to the operation. The register operation returns a RegistrationResponse object. This object is populated with information relative to the operation's response. The two main items of interest in the RegistrationResponse object are the registrationId and statusCode property. The registrationId property contains necessary registration data. If the statusCode is less than zero, an error has occurred and then more information about the error can be found in the errorMsg property.

   Input: RegistrationRequest
   Output: RegistrationResponse

2. **getTransaction**

   This operation returns a TransactionDetailResponse object populated with information based on the requested transactionId that was originally registered with WPS. After the TransactionDetailResponse object is obtained, the statusCode property should be analyzed. A statusCode of less than zero, an error has occurred and errorMsg is populated with more information.

   Input: registrationId
   Output: TransactionDetailResponse

3. **getStatus**

   This operation returns some basic status information about the registration that was originally registered with WPS. A StatusResponse is returned to the user as a result of this operation. After a StatusResponse object is obtained, the statusCode should be analyzed. A statusCode of less than zero indicates an error has occurred and the errorMsg of the StatusResponse object has

more information about the error that occurred. The getStatus operation is useful when a particular transaction is registered with the expirationTime property being greater than zero.

Input: registrationId
Output: StatusResponse

Protocol level properties:

USERNAME_PROPERTY which is the username.
PASSWORD_PROPERTY which is the password.
ENDPOINT_ADDRESS_PROPERTY which is the web service endpoint.

Note the protocol level properties needed to be set because the WPS web service uses BASIC authentication for authenticating and authorizing usage of this web service. See the example code on how to set this for JAX-WS.

WSDL URL for Test:
https://test.secure.utah.gov/wpsv2/WpsService?wsdl

WSDL URL for Production:
https://secure.utah.gov/wpsv2/WpsService?wsdl

Operation Name	*Input*	*Output*	*Faults*
register	* WpsAccount * RegistrationRequest	RegistrationResponse	Generic SOAP fault should an error occur
getTransactions	* WpsAccount * RegistrationID	TransactionDetailResponse	Generic SOAP fault should an error occur
getStatus	* WpsAccount * RegistrationID	StatusResponse	Generic SOAP fault should an error occur

CREDIT CARD SECURITY

Utah GovPay has been designed to protect consumer's sensitive credit card information from theft through strong security procedures. The procedures have also been designed to reduce the risk and liability to a state agency in processing and handling credit cards. They have also been designed to reduce the liability for the State of Utah to be compliant with the Data Security Standard imposed by the credit card companies.

The procedures used by Utah GovPay for handling and storing credit card information is described in the following steps.

Step 1. The consumer is handed from the state agency's application to Utah GovPay before any credit card information is requested or entered. This step limits the credit card information to be only handled between the consumer's browser and the SSL connection to Utah Interactive, which is outside of the firewall that the State of Utah uses to protect agency web applications.

Step 2. After the consumer is connected to the Utah Interactive, they are asked to enter in their credit card number and address. The credit card information is encrypted and sent to the Utah GovPay website where the credit card information is then unencrypted inside the Utah Interactive firewall.

Step 3. The transaction details are logged in a Database but the full credit card number, credit card expiration date, and the CVV code are not included as part of this log. Only the first 2 digits and last 4 digits of the credit card number are stored in the log. The CVV code and the expiration date are never stored in the database log file.

Step 4. The full payment details, including the full credit card number are sent to a third party processor such as Paymentech. These details are encrypted with SSL inside the Utah Interactive firewall and transmitted directly to the payment processor. A unique transaction number created by Utah Interactive is also sent with the credit card information. This transaction happens outside of the firewall provided by the State of Utah.

Step 5. The payment processor responds to Utah GovPay with the transaction number and a message that determines if the transaction was accepted or declined. No credit card information is sent in this response message. This message is also sent via SSL encryption.

Step 6. Utah GovPay logs the payment results in the database. The log does not include full credit card numbers, expiration date, or CVV code. At this point, Utah Interactive does not store any record of the full credit card account number, expiration date or CVV code.

Step 7. Utah GovPay responds to the consumer that the transaction was either accepted or declined. The user is able to try again and it is recorded in the Utah GovPay as a separate payment transaction.

Step 8. The consumer then is either directed back to the state agency website or completes the transaction. The state agency can query the results of the transaction but the credit card number, expiration date and CVV codes are never sent to the agency.

Utah GovPay has a reporting website where state employees are able to log in and see the details of each payment attempt. The Utah GovPay reports displays the following information:

- Credit card type,
- Partial credit card Number,
- Name on credit card,
- Address,
- Transaction ID,
- Transaction Status Messages,
- Date & Time.

Throughout the entire payment transaction, the consumer's sensitive payment information such as full credit card number, expiration date, and CVV are never stored on in any record in Utah GovPay and are never transmitted into the state's firewall. Every time sensitive payment information is sent over the Internet between the consumer and Utah Interactive or between Utah Interactive and the third party payment processor it is encrypted with SSL.

Complex XML Types

RegistrationRequest				
Property	**Type**	**Size**	**Required**	**Description**
allowedPaymentTypes	String []	128	Yes	An array of values indicating the types of payment a user can make. Possible values: CREDITCARD, ECHECK.
items	RequestItem []	> 0	Yes	An array of RequestItem objects.
addrLine1	String	128	No	Value to use to pre-populate the credit card address line 1 address.
addrLine2	String	128	No	Value to use to pre-populate the credit card address line 2 address.
city	String	128	No	Value to use to pre-populate the credit card city field.
emailAddr	String	128	No	Value to use to pre-populate the credit card email address field.
name	String	128	No	Value to use to pre-populate the credit card name field.
postalCode	String	128	No	Value to use to pre-populate the credit card postal code field.
stateProvince	String	128	No	Value to use to pre-populate the credit card state/province field.

sharedSecretName	String	128	eChecks	The name of the shared secret to display to the user. eChecks require users to confirm some information.
sharedSecretValue	String	128	eChecks	A value that the user should know that is used to authenticate them when making eCheck payment.
successMsg	String	128	No	A message to be displayed upon successful payment.
successUrl	String	No Limit	No	The URL where the user is sent upon a successful payment.
failUrl	String	No Limit	No	The URL where the user is sent upon an unsuccessful payment.
expirationTime	integer	> 0	No	The number of minutes to allow this transaction to remain active in WPS before the transaction times out. The default is 0 and is interpreted as no timeout. In other words, if this value is not specified, the transaction will never timeout.

RequestItem

Property	Type	Size	Required	Description
amountEach	double	> 0	Yes	The dollar amount that this RequestItem costs.
customerId	String	128	Yes	A value that uniquely identifies the customer in the calling application. Examples include license or account numbers.
customFields	RequestItemCustomField []	N/A	No	An array of custom fields that are passed into WPS. See the reference for RequestItemCustomField.
description	String	255	Yes	A description of the item.
quantity	double	> 0	Yes	The quantity of this item.
transactionId	String	128	Yes	Unique identifier for the transaction in the calling application. Must be unique.
transactionType	String	128	No	A code identifying the type of transaction that this item is participating in. If applicable, the FINET code should be put here.

RequestItemCustomField				
Property	**Type**	**Size**	**Required**	**Description**
name	String	64	Required only when value is also specified	The name of the custom field
value	String	128	Required only when name is also specified	The value of the custom field

RegistrationResponse				
Property	**Type**	**Size**	**Required**	**Description**
statusCode	integer	> 0	Yes	The status code indicates the status of the transaction. If status code is less than 0, it indicates there was a problem with the request and greater than 0 indicates the request succeeded. If status code is less than 0, the error message should be analyzed for further information.
errorMsg	String	No limit	No	This will only be populated if the status code has a value less than zero which indicates there was a problem completing the users request.
registrationId	String	128	No	The registration id of the transaction that was registered with WPS.

StatusResponse				
Property	**Type**	**Size**	**Required**	**Description**
statusCode	integer	> 0	No	The status code indicates the status of the transaction. If status code is less than 0, it indicates there was a problem with the request and greater than 0 indicates the request succeeded. If status code is less than 0, the error message should be analyzed for further information.
errorMsg	String	No limit	No	This will only be populated if the status code has a value less than zero which indicates there was a problem completing the users request.
transactionStatus	String	No limit	No	The following are the statuses and their meanings: timed-out=the registered transaction has exceeded its time limit in WPS. error=the payment for this transaction received an error at the payment gateway. successful=the transaction was successful declined=the payment was declined for this transaction not-found=the requested transaction could not be found (the registration id is invalid)
isTimedOut	boolean	N/A	Yes	This is set to false by default. It is only true if the registered request has exceeded the timeout period set in the expirationTime property located in the RegistrationRequest object.

TransactionDetailResponse				
Property	Type	Size	Required	Description
statusCode	integer	> 0	Yes	The status code indicates the status of the transaction. If status code is less than 0, it indicates there was a problem with the request and greater than 0 indicates the request succeeded. If status code is less than 0, the error message should be analyzed for further information.
errorMsg	String	No limit	No	This will only be populated if the status code has a value less than zero which indicates there was a problem completing the users request.
registrationId	String	128	No	The registration id of the transaction that was originally registered with WPS.
addrLine1	String	128	No	The first address line of the transaction that was originally registered with WPS.
addrLine2	String	128	No	The second address line of the transaction that was originally registered with WPS.
authorizationCode	String	10	No	The code provided by the payment processor.
auxiliaryMessage	String	255	No	A message that provides further information about the status message.
city	String	128	No	City used for the payment.
completionDate	Timestamp	N/A	No	The date the transaction completed.
country	String	128	No	The country used for payment.
error	String	10	No	"true" or "false" The false result could be caused by either a success or a declined transaction.
gatewayTransactionId	String	128	No	The transaction id provided by the payment gateway.
name	String	128	No	Name used for payment.

orderId	String	16	No	Unique order id automatically assigned by the system.
paymentSuccessful	String	10	No	"true" or "false" The false result could be caused by either a technical error or a declined transaction.
postalCode	String	10	No	The postal code used for the payment.
stateProvince	String	2	No	The 2 character state/province code used for the payment.
statusMessage	String	255	No	The message provided by the payment processor.

Note: The total payment amount for the transaction is calculated by multiplying each wps.service.RequestItem quantity by the wps.service.RequestItem's amount.

Web Service JAX-WS example

The following code is a sample implementation of how to use a JAX-WS client to connect to our WPS web service. Please note that the WpsService and WpsServiceService are generated by the wsgen generator.

```
package mywpstest;
import javax.xml.ws.BindingProvider;
/**
 *
 * @author jkilgrow
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)    {
        Main main = new Main();

        WpsService service = new WpsService(
                new URL("https://test.secure.utah.gov/wpsv2/WpsService?wsdl"),
                new QName("http://service/", "WpsService"));
        Wps client = service.getWpsPort();
        BindingProvider provider = (BindingProvider) client;
        provider.getRequestContext().put(BindingProvider.USERNAME_PROPERTY, "junit_test");
        provider.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, "+49egacr");
        provider.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
"https://test.secure.utah.gov/wpsv2/WpsService?wsdl");

        RegistrationResponse response = main.register(client);

//          now we can examine the response to see if our registration was successful or not.
        if (response.getStatusCode() < 0)       {
            System.out.println("An error occurred during transaction registration. The error message is: " +
response.getErrorMsg());
            System.exit(response.getStatusCode());
        }

        System.out.println("Apparently the transaction was successfully registered");
        System.out.println("Registration ID: " + response.getRegistrationId());


        // get the status of the registered transaction
        StatusResponse status = client.getStatus(response.getRegistrationId());

        if (status.getStatusCode() < 0)       {
            System.out.println("there was a problem getting the status");
            System.out.println("error msg: " + status.getErrorMsg());
        }

        System.out.println("status response: " + status.getTransactionStatus());


        // get the transaction
        TransactionDetailResponse transaction = client.getTransaction(response.getRegistrationId());
```

```
        if (transaction.getStatusCode() < 0)        {
            System.out.println("there was a problem getting the transaction");
            System.out.println("error msg: " + transaction.getErrorMsg());
        }

        System.out.println("transaction response: " + transaction.getStatusMessage());
    }

    private RegistrationResponse register(Wps client)    {
        RegistrationRequest request = new RegistrationRequest();
        request.getAllowedPaymentTypes().add("CREDITCARD");
        request.getAllowedPaymentTypes().add("ECHECK");
        request.setAddrLine1("123 Main St");
        request.setCity("Salt Lake City");
        request.setStateProvince("UT");
        request.setEmailAddr("test@test.com");
        request.setName("Test Request 1");
        request.setPostalCode("84111");
        request.setExpirationTime(0);  // initially, we'll try the default timeout period
        RequestItem item = new RequestItem();
        item.setAmountEach(5);
        item.setCustomerId("abc123");
        item.setDescription("transaction item 1");
        item.setQuantity(1);
        item.setTransactionId("abc123-1");
        request.getItems().add(item);   // potential failure point. what if the result of getItems() is null?
        RegistrationResponse response = client.register(request);
        return response;
    }
}
```